Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Curso de PHP Avanzado

Programación de sitios web dinámicos
PHP - JavaScript - MySQL

Primera Parte

Coordinación: **Carlos Rico**

Autor: **Leonardo Tadei**

Marzo 2013

Software:

Existen varias definiciones similares aceptadas para software, pero probablemente la más formal sea la siguiente:

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Extraído del estándar 729 del IEEE

Considerando esta definición, el concepto de software va más allá de los programas de computación en sus distintos estados: código fuente, binario o ejecutable; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software: es decir, abarca todo lo intangible, todo lo "no físico" relacionado.

Ahora bien, para crear un software, existen varias técnicas y procedimientos que son exclusivos de esta actividad, y que nos ayudan a llegar el resultado final, es decir, nuestro producto de software, independientemente de que sea un software para la web, para PCs de escritorio, para teléfonos móviles, para servidores, centrales telefónicas o automóviles.

Siguiendo estas pautas, no se garantiza que el software funcione bien o que resuelva el problema para el cual había sido creado, pero no seguir estas pautas, reduce mucho las posibilidades de terminar bien, es decir, con un software correcto, completo, sin fallos, entregado a tiempo y dentro del presupuesto estimado.

Proceso de creación del software

Se define como Proceso al conjunto ordenado de pasos a seguir para llegar a la solución de un problema u obtención de un producto, en este caso particular, para lograr la obtención de un producto software que resuelva un problema.

El proceso de creación de software puede llegar a ser muy complejo, dependiendo de su porte, características y criticidad del mismo. Por ejemplo la creación de un sistema operativo es una tarea que requiere proyecto, gestión, numerosos recursos y todo un equipo disciplinado de trabajo. En el otro extremo, si se trata de un sencillo programa (por ejemplo, la resolución de una ecuación de segundo orden), éste puede ser realizado por un solo programador (incluso aficionado) fácilmente. Es así que normalmente se dividen en tres categorías según su tamaño (líneas de código) o costo: de Pequeño, Mediano y Gran porte.

Se estima que, del total de proyectos software emprendidos, un 28% fracasan, un 46% caen en severas modificaciones que lo retrasan y un 26% son totalmente exitosos.

Cuando un proyecto fracasa, rara vez es debido a fallas técnicas, la principal causa de fallos y fracasos es la falta de aplicación de una buena metodología o proceso de desarrollo. Entre otras, una fuerte tendencia, desde hace pocas décadas, es mejorar las metodologías o procesos de desarrollo, o crear nuevas y concientizar a los profesionales en su utilización adecuada.

Es común para el desarrollo de software de mediano porte que los equipos humanos involucrados apliquen sus propias metodologías, normalmente un híbrido de los procesos habituales y a veces con criterios propios.

El proceso de desarrollo puede involucrar numerosas y variadas tareas, desde lo administrativo, pasando por lo técnico y hasta la gestión y el gerenciamiento. Pero casi rigurosamente siempre se cumplen ciertas etapas mínimas; las que se pueden resumir como sigue:

* Captura, Elicitación , Especificación y Análisis de Requerimientos (SRS)
* Diseño
* Codificación
* Pruebas (unitarias y de integración)
* Instalación y paso a Producción
* Mantenimiento

En las anteriores etapas pueden variar ligeramente sus nombres, o ser más globales, o contrariamente, ser

más refinadas.

Pero independientemente del tamaño del software, la primer etapa de Elicitación de Requerimientos es necesaria, ya que no podremos escribir un software son saber antes qué software debemos escribir, y daremos aquí unas breves pautas aplicadas a proyectos web típicos que usaremos durante todo este curso.

Gestión de Proyectos de Software

Una de las principales incertidumbres a la hora de querer trabajar construyendo aplicaciones web y software en general la produce el no saber por dónde empezar. En esta sección vamos a tratar de esclarecer aunque superficialmente cómo abordar un proyecto de software.

Para esto, vamos a comenzar con la definición de Proyecto: "*Se puede definir PROYECTO como un conjunto de actividades interdependientes orientadas a un fin específico, con una duración predeterminada*".

Esto nos divide el problema en 3 partes que podremos abordar por separado:

determinar la finalidad del proyecto: parece obvio, pero la mayoría de los proyectos fracasan porque no está claramente determinado cuál es el resultado final al que se debe llegar. Esto convierte a la etapa de especificación de requerimientos en un paso muy importante, ya que el éxito o el fracaso de un proyecto se determina viendo si cumplió o no con su objetivo, para lo cual necesitamos primero conocer nosotros y el interesado en el software perfectamente ese objetivo. El requerimiento debe volcarse por escrito, y hasta se suele firmar por las partes a modo de contrato.

determinar cuales serán las actividades independientes: estas actividades pueden variar de proyecto en proyecto, pero las que siempre estarán presentes serán a) saber qué hay que hacer (requerimiento), b) armar un modelo de datos válido, c) diseñar los procesos que manipularán los datos, d) diseñar las interfaces con que el usuario usará los procesos y visualizará los datos (en esta etapa pueden surgir nuevos procesos "de interfaz"), e) probar que el sistema cumpla con el requerimiento determinado al principio.

calcular la duración: una vez determinadas las actividades independientes recién podrá estimarse la duración del proyecto. En este cálculo los informáticos tendemos a ser muy optimistas y consideramos menos tiempo del que realmente necesitamos, no por no saber calcular el tiempo en hacer una actividad, sino por no considerar factores ajenos que nos impiden dedicar todo nuestro tiempo al desarrollo. Es una buena política sumar a los proyectos entre un 10 y un 25% del tiempo calculado optimistamente hasta ser capaces de incluir estas demoras en el propio cálculo. Naturalmente que el cálculo del tiempo es una parte importante del cálculo de costos de un sistema, junto con su complejidad, arquitectura, restricciones de hardware, etc.

Especificación de Requerimientos de Software (SRS)

Una especificación de requerimientos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requerimientos no funcionales (o suplementarios). Los requerimientos no funcionales son aquellos que imponen restricciones al diseño o funcionamiento del sistema (tal como requerimientos de funcionamiento, estándares de calidad, o requerimientos del diseño).

Las estrategias recomendadas para la especificación de los requerimientos del software están descritas por IEEE 830-1998. Este estándar describe la estructuras posibles, contenido deseable, y calidades de una especificación de requerimientos del software.

Los requerimientos pueden ser de tres tipos:

* Requerimientos Funcionales: son los que el usuario necesita que efectúe el software. Ej: el sistema debe emitir un comprobante al registrar la entrega de mercadería.
* Requerimientos No funcionales: son los "recursos" para que trabaje el sistema de información (redes, tecnología).Ej: el soporte de almacenamiento a usar debe ser MySQL
* Requerimientos Empresariales u Organizacionales: son el marco contextual en el cual se implantará el sistema para conseguir un objetivo macro. Ej: abaratar costos de expedición.

Una correcta Especificación de Requerimientos de Software produce requerimientos:

- organizados
- medibles
- comprobables
- expresados con lenguaje mínimo
- sin ambigüedades o contradicciones
- ranqueables
- homogéneos

La especificación de requerimientos es un "documento vivo" que se usa durante todo el ciclo de desarrollo del sistema y muchas veces cambia y se actualiza, por tanto para su mejor uso, debe tener una portada con el nombre del proyecto, los autores y la versión de la especificación.

Definición de requerimientos funcionales:

La definición de requerimientos funcionales consiste en la caracterización de lo que el sistema (artefacto) debe hacer. Debe hacerse referencia exclusivamente a lo que el sistema debe hacer, y nunca a lo que el usuario hará con el sistema: estamos caracterizando al sistema, no a quienes lo usarán.

Cada requerimiento debe estar numerado, de forma tal que pueda hacerse referencia a él desde otro requerimiento, y ordenado de forma tal que cuando un requerimiento mencione a otro, este otro ya haya sido definido.

Cada requerimiento debe definir una funcionalidad y solo una funcionalidad del sistema, ya que de otra manera se pierde la homogeneidad de la SRS. En caso de que un requerimiento sea demasiado extenso, lo que significa que su definición puede ser demasiado general, se subdivide en items, en los cuales se redactarán los subrequerimientos que lo componen.

Dado que lo que debemos hacer es definir qué hace el sistema, y no por ejemplo qué es lo que los usuarios harán con el sistema, cada requerimiento funcional comienza con "El sistema debe..." y luego lo que hará el sistema. Ej: El sistema debe mostrar el saldo de cuenta corriente de los clientes.

Es por esto que, y aunque parezca extraño a primera vista, en la Especificación de Requerimientos Funcionales, el usuario jamás es nombrado, ya que la lista de funcionalidades que el sistema debe cumplir no depende del usuario que la usa (si bien podrá darse el caso de que un usuario no pueda acceder a ella, de todos formas el sistema sí debe implementarla).

Por una cuestión de simplicidad, convenimos en usar la palabra "gestión" como un sinónimo de Presentación, Altas, Bajas y Modificaciones de los datos. Por Ej: "El sistema debe gestionar clientes" es una abreviatura de cuatro requerimientos funcionales, a saber "El sistema debe mostrar clientes", "El sistema debe dar de alta clientes", "El sistema debe dar de baja clientes" y "El sistema debe modificar clientes".

La Especificación de Requerimientos Funcionales tiene generalmente dos apéndices. Uno es el Diccionario, en dónde se definen los términos ambiguos o desconocidos usados en la redacción de los requerimientos y se los acota al contexto del sistema. Por ejemplo un "vendedor" no es lo mismo para un sistema de ventas en un comercio, que para uno de una compañía de seguros, que para una empresa que envía vendedores a recorrer ciudades y tomar pedidos. En el Diccionario es el lugar ideal para, al definir el término, enumerar los atributos que los caracterizan, como por ejemplo:

Vendedor: persona que recorre en un auto propio pueblos y ciudades y toma pedidos a los Clientes de la Empresa, a la vez que intenta sumar nuevos. Atributos:
Nombre
Apellido
Tipo y Nro de Documento.
Teléfono
Sueldo base
Comisión por venta (%)
Cuota de ventas ($)
Premio por cuota cumplida ($)

El otro apéndice usado es el que tiene un modelo de los listados a emitir por el sistema, detallando el título, las columnas del listado y los totalizadores. Por cada Requerimiento Funcional que implique emitir un

listado, debería haber un modelo de la salida impresa a generar. Es habitual hacerlos en hojas de cálculo, para documentar también las fórmulas que se usan para los totalizadores.

Una Especificación de Requerimientos Funcionales detallada, puede ser usada como un contrato informal entre el cliente que pide el desarrollo del sistema y los que lo construyen, ya que ahí estás enumeradas cada una de las funcionalidad a cumplir, con lo que conseguimos determinar claramente se un pedido del cliente es una omisión nuestra, de la cuál deberemos hacernos cargo, o un olvido del cliente, en cuyo caso deberemos cotizar la ampliación del sistema para que sea aceptada. Dado que el producto de nuestro trabajo es intangible, ser muy claros siempre es una cuestión importante.

Cómo funciona la World Wide Web:

Funcionamiento de un Web Site:

El funcionamiento de un Web-Site es un ejemplo típico de arquitectura de servicio, en donde múltiples clientes se conectan a un servidor (en algunos casos varios) en forma simultanea. En general el servidor depende de la instalación del sitio mientras que el cliente suele ser un navegador web (browser), por ejemplo Mozilla Firefox, Iceweasel, Opera, Safari, Internet Explorer, Galeon, Lynx, Chrome, etc. Como en todo esquema de servicio debe existir un protocolo que especifique de que forma se comunican e intercambian datos el cliente y el servidor, el protocolo utilizado en un web site es el protocolo HTTP que funciona “encapsulado” sobre el protocolo TCP/IP.

Introducción al Protocolo HTTP:

Sus siglas significan Hyper Text Transfer Protocol (Protocolo de Transferencia de HiperTexto). Básicamente el protocolo es iniciado por el cliente con un “request”, es decir un pedido de un recurso determinado, que es casi siempre contestado por el server con el envío de una respuesta (“response”) que incluye un código indicando si el pedido pudo ser resuelto por el server o no.
Un request genérico tiene la forma:

METODO URI PROTOCOLO CrLf
HEADERS*
CrLf
Datos

El MÉTODO en general puede ser GET o POST
URI es el identificador del recurso que se desea pedir, el formato es: http://host:port/path?query_string
PROTOCOLO debe ser HTTP / 1.1
CrLf es un Carriage Return seguido de un New Line (0x13,0x10)
Headers son de tipo: Header-Name: Value CrLf, y pueden indicar varias cosas.

Dado que el protocolo es "de texto", todo lo que se transmite entre el cliente y el servidor es simplemente texto, con comandos específicos que indican qué es lo que el cliente pide. La respuesta también es texto, que es analizados por el navegador para visualizarlo decorado como los tags lo indican.

Un ejemplo de pedido usando un cliente TELNET es:

```
$ telnet intranet.com 80
Trying 192.168.0.99...
Connected to intranet.com.
Escape character is '^]'.
GET / HTTP/1.1 <Enter>
Host: intranet.com <Enter>
<Enter>
```

Nótese que la página solicitada es "/", es decir, se pide por la raíz del sitio, y que se especifica la versión del protocolo, en este caso la 1,1 y que se indica el "Host" o nombre de dominio requerido, ya que este es un requisito de la versión 1,1 del protocolo que de esta manera permite que existan varios dominios en una misma dirección IP de servidor.

El servidor distingue que la petición ha terminado cuando detecta una línea en blanco, representada

en el ejemplo con el <Enter> vacío.

Un ejemplo de respuesta de un servidor podría ser:

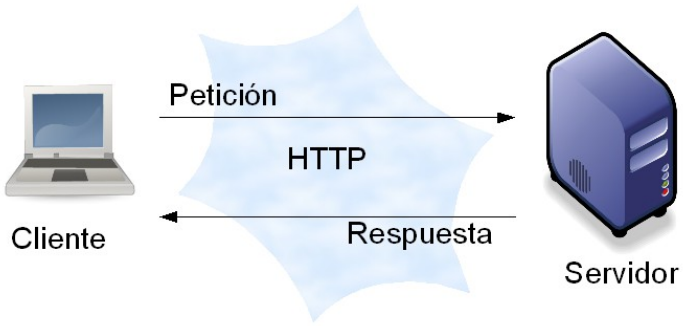
```
HTTP/1.1 200 OK
Date: Thu, 25 Mar 2010 04:25:01 GMT
Server: Apache/2.2.4 (Debian) PHP/5.2.6-1+lenny3 with Suhosin-Patch
Last-Modified: Wed, 22 May 2002 21:23:59 GMT
ETag: "e823c-24f-d0b8b9c0"
Accept-Ranges: bytes
Content-Length: 591
Content-Type: text/html

<html>
Datos…..
```

Nótese como otra vez, una línea en blanco distingue la cabecera de la respuesta del principio del contenido a mostrar al usuario.

Es importante destacar que en ambos casos es necesario enviar dos <enter> para indicar que el pedido termina. De la misma manera, el servidor separará con dos <enter> la cabecera de la respuesta del contenido, que en este caso es un archivo HTML.

Los datos que el servidor envía al browser (navegador) dependen del “Content-Type” declarado, básicamente los tipos más usados son texto plano (text/plain), código html (text/html), o imágenes (image/gif u otros).
De esta forma el cliente y el servidor se comunican por medio de tantos ciclos REQUEST-RESPONSE como sean necesarios, es de destacar que por cada REQUEST se debe iniciar una conexión nueva entre el cliente y el servidor ya que el protocolo HTTP en su forma básica no admite que en una misma conexión se haga más de un pedido al servidor. En una página html simple con 3 imágenes por ejemplo es normal que se efectúen 4 conexiones al servidor: una para la página y luego una por cada imagen.



Tecnologías disponibles para el desarrollo de aplicaciones:
Para desarrollar aplicaciones y dotar a las páginas web de funcionalidad se puede trabajar tanto en el lado del cliente como en el lado del servidor, las variantes son:

Programación en el cliente:
· El browser envía un request.
· El server envía un response que contiene código que el browser entiende.
· El browser interpreta el código enviado por el server y realiza una determinada acción.

Programación en el servidor:
· El browser envía un request.
· El server ejecuta una aplicación que realiza una determinada acción.
· El server envía el resultado de dicha aplicación al cliente.
· El browser muestra el resultado recibido del server.

Esquema mixto: (programación en el cliente y en el servidor)
· El browser envía un request.
· El server ejecuta una aplicación que realiza una determinada acción.
· El server envía el resultado de dicha aplicación al cliente conteniendo código a interpretar por el browser.
· El browser interpreta el código enviado por el server y realiza una determinada acción.

La programación del lado del cliente tiene como principal ventaja que la ejecución de la aplicación se delega al cliente, con lo cual se evita recargar al servidor de trabajo. El servidor sólo envía el código, y es tarea del browser interpretarlo. La gran desventaja de esta metodología es que el código que el server envía es "sensible" a que cosas puede o no hacer el browser. El usuario puede, por ejemplo, decidir deshabilitar una funcionalidad del browser que es necesaria para que se ejecute un determinado servicio o peor aún, browsers distintos pueden interpretar el mismo código de distintas formas. Típicamente Netscape y Microsoft, que producían los dos browser más usados del mercado, no se ponen de acuerdo sobre como se implementan diversas tecnologías en el cliente.
Programar del lado del servidor tiene como gran ventaja que cualquier cosa puede hacerse sin tener en cuenta el tipo de cliente, ya que la aplicación se ejecuta en el servidor que es un ambiente controlado. Una vez ejecutada la aplicación, el resultado que se envía al cliente puede estar en un formato "normalizado" que cualquier cliente puede mostrar. La desventaja reside en que el server se sobrecarga de trabajo ya que además de servir páginas es responsable de ejecutar aplicaciones. A menudo esto redunda en requisitos de hardware mayores a medida que el server ejecuta más y más servicios.

Programación en el cliente	**Programación en el servidor**
HTML CSS DHTML JavaScript Java (applets) VBScript	CGI (Cualquier Lenguaje) ASP PHP mod_perl Java (servlets) mod_python

Debido a las incompatibilidades existentes y a la posibilidad de que el usuario controle que cosas se ejecutan y cuales no la programación del lado del cliente no es muy recomendable y debe limitarse a código altamente standard que pueda interpretarse de cualquier forma en cualquier browser, lo cual obliga a ejecutar la gran mayoría de las aplicaciones y servicios de un web site del lado del servidor.

Server Side Programming.

Para el desarrollo de aplicaciones del lado del servidor existen 3 grandes metodologías, utilizar el protocolo CGI, utilizar una API provista por el web-server o bien utilizar un "módulo" del web server.

El protocolo CGI:
El protocolo CGI (Common Gateway Interface) fue creado para establecer un protocolo standard de comunicación entre el web-server y cualquier lenguaje de programación de forma tal que desde el lenguaje "x" puedan recibirse datos que el usuario envía usando el método "POST" o "GET" y además el resultado de la aplicación sea derivado por el web-server al browser. Típicamente para recibir datos se usa alguna biblioteca o módulo del lenguaje elegido que implementa el protocolo CGI y para enviar datos simplemente se envían al standard-output desde el lenguaje elegido y el web-server se encarga de redireccionar esto al browser.
Para ejecutar una aplicación CGI el web-server en general procede de la siguiente manera:
· Se toma el "request del browser" y los datos que se envían al server por método "GET" o "POST" se pasan a variables de ambiente.
· El server redirecciona su salida standard al browser.
· El server crea un proceso (Fork) (que tiene la salida standard redireccionada)
· El server ejecuta en el proceso creado la aplicación deseada.
· Se ejecuta la aplicación

Cuando la aplicación termina de ejecutarse el proceso muere. Dentro de la aplicación se usa algún mecanismo para recuperar los datos enviados por el browser desde las variables de ambiente (todos los lenguajes manipulan variables de ambiente). El protocolo CGI justamente consiste en especificar de que forma los datos enviados por el browser se convierten en variables de ambiente, esto en general es transparente al usuario.
De esta forma pueden realizarse aplicaciones para un web-site en casi cualquier lenguaje, los lenguajes interpretados rápidamente ganaron terreno ya que tienen un ciclo de desarrollo en tiempo inferior a los lenguajes compilados y son más fáciles de debuggear dentro del ambiente CGI.
Los lenguajes no interpretados (C, C++) tienen como ventaja que requieren menos recursos del server al generarse el proceso CGI (no hace falta un interprete) y además suelen ser mucho más veloces en su

ejecución (no se necesita interpretar nada), sin embargo el desarrollar y debuggear suelen ser tareas muy complejas y no siempre se justifica el esfuerzo si la aplicación es pequeña. En los comienzos de la web la gran mayoría de las aplicaciones se encontraban en la categoría chica / muy chica por lo que la eficiencia no era un factor importante y por eso los lenguajes compilados no se utilizaron demasiado.
La desventaja de las aplicaciones CGI consiste en que el server debe realizar un fork, y ejecutar la aplicación o bien el interprete de la aplicación, y este ciclo que se cumple cada vez que se ejecuta la aplicación CGI insume muchos recursos y en general es costoso en tiempo para el server. Durante muchos años este esquema no muy eficiente dominó ampliamente el mundo de las aplicaciones Web.

Uso de una API del servidor:
Otra técnica factible consiste en utilizar una API (application programming interface) provista por el webserver para desarrollar aplicaciones, es decir que el web-server provee un lenguaje en el cual se pueden desarrollar aplicaciones. Este esquema, como podemos apreciar, es mucho más eficiente que el anterior ya que el web-server es el encargado de ejecutar las aplicaciones en forma directa sin necesidad de crear un proceso. Las desventajas son sin embargo importantes: en primer lugar las aplicaciones creadas en este marco no son portables ya que sólo pueden ejecutarse en un web-server determinado, esto es una gran desventaja frente a las aplicaciones CGI que podían una vez desarrolladas ejecutarse en cualquier servidor. La segunda gran desventaja es que frecuentemente un error de programación de una aplicación podría ocasionar que el web-server deje de funcionar, genere un error, se cuelgue, pierda memoria u otros problemas. Esto ocasiona que este tipo de aplicación no sea confiable.

Uso de un "módulo del web-server"
La tecnología más reciente para la ejecución de aplicaciones consiste en anexar a un web-server "módulos" que permiten al web-server interpretar un determinado lenguaje. De esta forma se logra eficiencia ya que el server no necesita crear un nuevo proceso por cada aplicación que ejecuta. Las aplicaciones son portables ya que son desarrolladas en un lenguaje standard que no depende del web-server, las aplicaciones son confiables ya que si bien pueden producir un error en el lenguaje en que están diseñadas si el módulo es sólido dichos errores no pueden comprometer al web-server. Al combinar las ventajas más importantes del ambiente CGI y un desarrollo basado en APIs evitando los inconvenientes de los mismos este esquema suele ser el más adecuado para ejecución de aplicaciones.

Cuadro Resumen:

	CGI (interpretado)	**CGI (compilado)**	**API del server**	**Modulo del server**
Ejemplos	Perl, Python	C, C++	Netscape Enterprise	PHP, ASP, Mod_perl, mod_python, FastCGI
Tiempo de desarrollo	Corto	Largo	Medio	Corto
Debugging	Sencillo	Complejo	Complejo	Sencillo
Confiabilidad	Alta	Alta	Baja	Alta
Eficiencia	Baja	Media	Alta	Alta

Benchmarks:

Test 1: 1000 ejecuciones de un programa de 1 línea.

Tecnología	**Tiempo (segundos)**
CGI : C	20,6
CGI : Perl	23,8
CGI : Python	45,2

PHP	16,2
Mod_python	30
Mod_perl	16,6
FastCGI	16,4

Test 2: 100000 ejecuciones de un programa de 1 línea.

Tecnología	Tiempo (segundos)
CGI : C	2522
CGI : Perl	2886
CGI : Python	5486
PHP	2420
Mod_python	3519
Mod_perl	2117
FastCGI	2120

Test 3: 10000 ejecuciones de un programa de 7000 líneas.

Tecnología	Tiempo (segundos)
CGI : C	258
CGI : Perl	963
CGI : Python	978
PHP	304
Mod_python	347
Mod_perl	476
FastCGI	280

En los tests puede verse como las variantes CGI insumen mucho tiempo de “lanzamiento” de la aplicación. Esto se prueba ejecutando muchas veces aplicaciones muy chicas (solo 1 línea de código). De esta forma podemos ver como en este aspecto las tecnologías que no requieren que el server genere un nuevo proceso (php, fastcgi, mod_perl, mod_python) son mucho más eficientes que aquellas que sí lo necesitan.
En el tercer test donde se ejecuta una aplicación de gran tamaño se puede apreciar que la variante “compilada” en CGI es muy eficiente ya que no requiere tiempo alguno de interpretación y el tiempo necesario para generar el proceso es mínimo en comparación con el tamaño de la aplicación. Lo importante de este tercer test es que tanto php, como mod_perl o fastcgi pese a requerir de un interprete son también veloces en este tercer test.

Evolución de los Web-Sites

Desde la aparición de la web distintos “modelos” o prototipos de web-sites fueron siendo predominantes en distintos periodos de tiempo, de acuerdo a las características “comunes” de los sitios predominantes en cada momento podemos diferenciar 3 generaciones de web-sites distintas:

Web-Sites de primera generación:
En un web-site de primera generación las páginas se desarrollaban, se subían al servidor y el servidor se encargaba de enviar las páginas al browser, es un modelo basado en páginas estáticas, en donde predominaba el uso de texto, links a otros sites o a otras páginas del mismo site y listas con “bullets” para enumerar cosas. Frecuentemente se usaban líneas horizontales “rules” para separar contenidos y las páginas de gran extensión vertical con gran cantidad de texto, listas y links eran comunes. Si bien no eran visualmente atractivos estos sites estaban enfocados a funcionar en forma veloz y entregar al usuario gran cantidad de información interrelacionada.

Web-Sites de segunda generación:
La segunda generación de web-sites implicó una revolución en lo visual, a medida que los web-sites se volvían emprendimientos más comerciales que científicos el hecho de “capturar” usuarios se torno una premisa y por ello se le dio gran importancia al aspecto visual. Las páginas con “layouts” visualmente atractivos fueron más y más populares, el uso abundante de imágenes, imágenes animadas y elementos multimedia se volvió común, aparecieron páginas que controlaban los fonts, el estilo y la posición en la que los mismos se ubicaban. La gran mayoría de los sites usaban tablas HTML para controlar la posición exacta en que los elementos aparecerían en el browser y tags nuevos en html como “font” u otros para controla la forma en la cual los mismos se verían. El concepto fue dominar la presentación de la información. Otra característica importante de esta segunda generación de Web-sites fue la aparición explosiva de más y más aplicaciones a medida. Los servicios que ofrecía un site se volvían factores importantes en la atracción de usuarios, chats, foros de discusión, banners, contadores y muchas aplicaciones más empezaron a aparecer y las falencias del protocolo CGI, a medida que las aplicaciones eran mas grandes y la cantidad de usuarios crecía exponencialmente, comenzaron a hacerse notar.

Web-Sites de tercera generación:
La tercera generación de web-sites siguió basada en lo visual, el gran cambio vino en la forma cómo se generaba la información. Las páginas estáticas que dominaban el 100% de los sitios de primera y segunda generación fueron reemplazadas por páginas dinámicas que el web-server generaba en el momento, a partir de información que en general se guardan en una base de datos. Estos sitios “dinámicos” permiten actualizar la información e incluso cambiar completamente la forma en que se muestran dichos datos en velocidades asombrosas. Los sitios de tercera generación facilitaron las aplicaciones interactivas, la información en tiempo real y que día a día se ofrecieran nuevos servicios. Las aplicaciones empezaron a desarrollarse también usando otras tecnologías dejando de lado el protocolo CGI. Aplicaciones en ASP, mod_perl o PHP, mucho más poderosas y eficientes que sus pares CGI, son el standard de este tipo de sitios.

Generalidades del lenguaje PHP

Introducción:
PHP es un lenguaje interpretado diseñado para favorecer el desarrollo de web-sites dinámicos y aplicaciones para web-sites. La distribución más popular de PHP es como módulo para el web-server Apache, aunque puede funcionar con las limitaciones que ya conocemos, como un interprete para ejecutar aplicaciones Cgi en aquellos web-servers que no lo soporten como módulo.
PHP se distribuye en formato open-source y es gratuito, una instalación habitual de PHP consiste en compilar el módulo PHP y luego recompilar el Apache para que utilice el módulo recientemente compilado.

Generalidades:
La característica más importante de PHP es que permite combinar código html y código php en una misma página (de extensión php), por ejemplo:

```
<?php print('<?xml version="1.0" encoding="UTF-8"?>'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head><title>Hola</title></head>
<body>
Hola esta es una prueba. <br>
<?php
print(“Hola soy una l&iacute;nea generada en php. <BR />”);
?>
</body>
```

```
</html>
```

Este ejemplo al guardarse en un archivo de extensión .php es automáticamente parseado [http://es.wikipedia.org/wiki/Parser] por el interprete de php cuando el browser envía un pedido. El ciclo es el siguiente:
· El browser envía un pedido de un archivo con extensión php.
· El server analiza que la extensión del request es .php, obtiene el archivo y lo envía al interprete php.
· El interprete php del web-server parsea el archivo en busca de tags <? ?> y procesa todo lo que se encuentre entre dichos tags (puede haber varias apariciones de los tags en un mismo archivo), todo aquello que esta fuera de los tags se envía al browser sin interpretar.
· El resultado combinado de aquello que no debe interpretarse y el resultado del código interpretado se envía al browser.
En nuestro ejemplo el browser recibiría:

```
<?php print('<?xml version="1.0" encoding="utf-8"?>'); ?>
<!doctype html public "-//w3c//dtd xhtml 1.1//en" "http://www.w3.org/tr/xhtml11/dtd/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head><title>hola</title></head>
<body>
Hola esta es una prueba. <br>
Hola soy una línea generada en php. <br>
</body>
</html>
```

Como podemos ver, es muy sencillo combinar código html y php. Para generar html desde php tenemos las siguientes opciones:
· Usar la función “print de php”
· Usar la función “echo de php”
· Cerrar el tag ?> escribir el código html deseado y volver a abrir el tag <?

La tercera opción es la más eficiente en velocidad cuando el código html que debemos generar es fijo, cuando el código html es dinámico podemos usar una mezcla de print y tags que abren y cierran que suele ser lo mas eficiente, por ejemplo:

```
<form name=<? print($nombre_form)?>>
etc
```

Introducción al lenguaje.

PHP [http://php.net] es un lenguaje ampliamente difundido, instalado en casi todos los servidores web existentes, sencillo de escribir y de leer y que cuenta con una gran cantidad de programadores y sistemas escritos en él. Según el índice TIOBE es actualmente el 3er lenguaje de programación más usado del mundo, y con una tendencia ascendente [http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html].
Se distribuye actualmente bajo licencia PHP License v3.01[http://www.php.net/license/3_01.txt], compatible con la Open Source Iniciative en vez de bajo una licencia dual con la GPL como originalmente sucedía. Su descarga y uso es gratuita, inclusive para usos comerciales.
Tiene interesantes características tales como poder conectarse con servidores de bases de datos (RDBMS), poder embeberse dentro de código HTML, soportar Programación Estructurada y Programación Orientada a Objetos, hacer casting automático [http://es.wikipedia.org/wiki/Conversión_de_tipos], asignar y librar memoria automáticamente, es multiplataforma, etc.
Técnicamente hablando PHP es un lenguaje interpretado, lo que significa que el código fuente se interpreta para su ejecución cada vez, en contraposición a los lenguajes compilados; es no tipado, por lo que no hace falta definir previamente los tipos de datos a usar; no posicional, por lo que no importa la columna en la cual se comience a escribir el código. Tampoco influye sobre el código la cantidad de saltos de línea (enter) que se coloquen, ni la cantidad de espacios.
La forma en la que se separan las distintas sentencias es mediante la utilización de “;”. En PHP cada sentencia debe finalizar con “;”.
Se puede escribir más de una sentencia en la misma línea siempre y cuando las mismas se encuentren separadas con “;”.

Comentarios:

En PHP hay 3 formas distintas de incluir comentarios:

```
/* Al estilo de C
en donde el comentario empieza
y termina delimitado por barra asterisco y asterisco barra
*/
```

O bien usando

```
// Comentario
```

O por último

```
# Comentario
```

En las dos últimas variantes el comentario empieza en donde se encuentra el "//" o el "#" y termina cuando termina la línea.

Tipos de Datos:

PHP soporta los siguientes tipos de datos:

- · Enteros
- · Binarios de punto flotante
- · Objetos
- · Vectores
- · Strings

En general el tipo de dato de una variable no es decidido por el programador sino que lo decide el lenguaje en tiempo de ejecución, la instrucción settype puede usarse para forzar el tipo de dato de una variable en los raros casos en que esto sea necesario. Todas las variables en php se denotan utilizando el signo '$' precediendo al nombre de la variable.

Enteros:
$a = 1234; # número decimal
$a = -123; # número negativo
$a = 0123; # número octal (83 en decimal)
$a = 0x12; #número en hexadecimal (18 decimal)

Flotantes:
Los números de punto flotante pueden notarse de la siguiente manera:
$a = 1.234;
$a = 1.2e3;

Strings:
En PHP los strings tienen un manejo similar al utilizado en "C" o "C++", están predefinidos los siguientes caracteres especiales:
\n Nueva línea
\r Salto de carro (carring return)
\t Tabulación
\\ Barra invertida
\$ Signo pesos
\" Comillas doble
Un string puede inicializarse usando comillas dobles o comillas simples. Cuando se utilizan comillas dobles el interprete de php parsea previamente el string, es decir que reemplaza los nombres de variables que encuentra en el string por el contenido de la variable. Cuando se usan comillas simples el string se imprime tal y como figura sin ser parseado.

Ej:
$x=”Juan”;
$s=”Hola $x”;
$t=’Hola $x’
$s vale “Hola Juan” y $t vale “Hola $x”.

Otra forma de inicializar un string es usando un string multilinea de la siguiente manera:

$str=<<<EOD
Este es un ejemplo de un string
que ocupa varias líneas y se puede
definir así
EOD;

Se pueden concatenar strings usando el operador “.” de la siguiente manera:
$x=”hola”;
$y=”mundo”;
$s=$x.$y; #$s es igual a “holamundo”.
$s=$x.” “.$y; # Aquí $s vale “hola mundo”

Vectores:
Los vectores en php actúan tanto como vectores tradicionales (indexados por número) así también como vectores asociativos (indexados por clave).
Los vectores pueden crearse usando las instrucciones “list” o “array” o bien inicializando en forma explícita cada elemento del vector.

$a[0]=”hola”
$a[1]=”mundo;
$a[“clave”]=”valor”;

Utilizando la notación especial $v[]; se pueden agregar elementos a un vector.

$a[0]=”nada”;
$a[1]=”hola”;
$a[]=”mundo”; #Asigna a $a[2] el valor “mundo”.

Existen funciones especiales para ordenar vectores, contar la cantidad de elementos de los mismos, recorrerlos, etc.

Matrices:
La definición, inicialización y uso de matrices en PHP es sencilla. Se puede pensar una matriz en PHP como un vector de vectores, por lo que se puede utilizar la misma lógica que en los primeros.

$a[0][1]=”Hola”;
$a[0][“clave”]=”una cosa”;
$a[“clave1”][“clave2”][0][1]=”otra cosa”;
etc…

Para incluir el valor de un elemento de un vector en un string se deben usar llaves para indicar el alcance del nombre de la variable a reemplazar:
print (“Esta es una prueba {$a[0][1]}”);

Una forma útil de inicializar un vector asociativo es usando la siguiente notación:

$a=array(
“color” => “rojo”,
“edad” => 23,
“nombre” => “juan”
);

Para crear una matriz se pueden anidar las declaraciones de tipo array.

```
$a = array(
"apple" => array(
"color" => "red",
"taste" => "sweet",
"shape" => "round"
),
"orange" => array(
"color" => "orange",
"taste" => "tart",
"shape" => "round"
),
"banana" => array(
"color" => "yellow",
"taste" => "paste-y",
"shape" => "banana-shaped"
)
);
```

Constantes:

Para definir una constante se utiliza la instrucción “define” de la forma:
define(“PI”,3.14151692);
Luego las constantes pueden usarse como variables tradicionales ($PI) con la salvedad de que no se les puede asignar un valor.

Operadores:

Los operadores aritméticos en PHP también se asemejan al C:

```
$a + $b; //suma
$a - $b; //resta
$a++; //pos-incremento, esta sentencia devuelve el valor de $a y lo incrementa en 1.
++$a; //pre-incremento, incrementa en 1 el valor de $a y devuelve el valor incrementado.
$a--; //pos-decremento
--$a; //pre-decremento
$a * $b; //multiplicación
$a / $b; //división
$a % $b; //módulo
```

Asignación:

La asignación se resuelve con el signo igual (“=”).

```
$a=5; //Asigna 5 a $a
$a=$b; //Asigna el valor de $b a $a
$b=($c=6); //Asigna 6 a $c y a $b
```

Y pueden combinarse asignación y operadores de la forma:

```
$a+=5; //Suma y asigna 5 a $a
$x.=”hola”; //Concatena $x con “hola”
```

Operaciones con bits:

```
$a & $b; //$a AND $b
$a | $b; //$a OR $b
$a ^ $b; //$a XOR $b
~$a; //NOT $a
$a << $b; //Shift hacia la izquierda $b posiciones
$a >> $b; //Shift hacia la derecha $b posiciones
```

Comparaciones:

$a = = $b; //true si $a igual a $b
$a = = = $b; //true si $a igual a $b y además son del mismo tipo
$a >= $b; //mayor o igual
$a <= $b; //menor o igual
$a != $b; //true si a y b son distintos
$a !== $b; //true si a y b son de distinto tipo aunque los valores sean semejantes

Operador @

Cuando se antepone @ a una expresión se suprimen los errores que la expresión pudiera generar.
Ej:
@($c=$a/$b);

Operador de ejecución:
PHP soporta el uso de “backticks” (comillas invertidas) para ejecutar un comando desde el shell y devolver el
resultado de la ejecución del comando en una variable:
$result=`ls –l`;

Operadores lógicos:

$a && $b; //True si $a es true y $b es true
$a || $b; //True si $a es true o $b es true
$a xor $b; //Or exclusivo
!$a; //True si $a es falso (NOT)

Estructuras de Control:

If:

```
if (expresión) sentencia;
if (expresión) {sentencias;}
if (expresión) {
   sentencias;
} else {
   sentencias;
}
if (expresión) {
   sentencias;
} elseif (expresión) {
   sentencias;
} else (expresión) {
   sentencias;
}
```

```
if ($foo):
    echo "sip\n";
elseif ($bar):
    echo "casi\n";
else:
    echo "nop\n";
endif;
```

```
$foobar = 2;
echo 'Foobar is ' . ($foobar == 2) ? 'foo' : 'bar';
// outputs 'foo';
echo 'Foobar is ' . (($foobar == 2) ? 'foo' : 'bar');
// outputs 'Foobar is foo';
```

While:

```
while (expresión) {
        sentencias;
}

while ($expresion):
        ...
endwhile;

do {
        sentencias;
} while(expresión)
```

For:

```
for (expr1,expr2,expr3) {
        sentencias;
}
```

La primera expresión cumple la función de inicializar las variables de control del FOR. Esta expresión se cumple incondicionalmente, más allá de que se entre dentro del ciclo o no.
La expresión 2 se evalúa siempre que se este por ingresar al ciclo del FOR, aún cuando se ingresa al for por primera vez.
La tercera expresión se ejecuta cada vez que se termina el ciclo. Por lo general se utiliza esta expresión para indicar el incremento de alguna variable que se este utilizando para el FOR. La ejecución de esta expresión es también incondicional y es la que se ejecuta inmediatamente antes de evaluarse la expresión 2.
Ejemplo:

```
for ($i=0;$i<5;$i++) {
   print("$i");
}
```

```
for (expr1; expr2; expr3): sentencia; ...; endfor;
```

Foreach:
Para realizar ciclos con la cantidad de ocurrencias en un vector se utiliza el comando foreach:

```
foreach ($vector as $variable) {
        sentencias;
}

foreach ($vector as $clave => $valor) {
        sentencias;
}
```

Por cada iteración cada elemento de $vector es asignado a $variable.
El segundo caso es aplicable para recorrer vectores asociativos.

Break:
Break permite salir del ciclo actual "for" , "while" o "switch"

Ejemplo:

```
$b=4;
for ($i=0;$i<6;$i++) {
   If($i==$b) break; //Sale del ciclo si $i es igual a $b
}
```

Switch:
El switch permite ejecutar un grupo de sentencias de acuerdo al valor de una variable:

```
switch ($variable) {
```

```
        case valor:
                sentencias;
                break;
        case valor2:
                sentencias;
                break;
        default:
                sentencias;
                break;
}
```

Cuando el valor de la variable ($variable) coincide con el valor de algún “case”, se ejecutan las sentencias que se encuentran a continuación.
En este caso se utiliza la sentencia “break” en forma prácticamente obligatoria, porque en caso de no existir esta sentencia se seguiría ejecutando linealmente todas las sentencias continuas, es decir las sentencias de los demás “cases” inferiores.
Por último, la opción “default” se utiliza generalmente para cuando el valor de la variable no coincide con ningún “case”. Estas sentencias se ejecutan siempre, salvo en el caso de que se ejecute antes un “break”.

Sintaxis Alternativa

PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, if, while, for, y switch. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:) y cerrar-llave por endif;, endwhile;, endfor;, or endswitch;, respectivamente.

```
<?php if ($a==5): ?>
 A es igual a 5
 <?php endif; ?>
```

En el ejemplo de arriba, el bloque HTML "A = 5" se anida dentro de una sentencia if escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si $a fuera igual a 5.

La sintaxis alternativa se aplica a else y también a elseif. La siguiente es una estructura if con elseif y else en el formato alternativo:

```
if ($a == 5):
    print "a es igual a 5";
    print "...";
 elseif ($a == 6):
    print "a es igual a 6";
    print "!!!";
 else:
    print "a no es ni 5 ni 6";
 endif;
```

Para el bucle While, los siguientes ejemplos son idénticos, y ambos imprimen números del 1 al 10:

```
/* ejemplo 1 */
 $i = 1;
 while ($i <= 10) {
    print $i++;  /* el valor impreso sería
                $i antes del incremento
                (post-incremento) */
 }

 /* ejemplo 2 */
 $i = 1;
 while ($i <= 10):
    print $i;
    $i++;
 endwhile;
```

Para el bucle for:

```
for (expr1; expr2; expr3):
   statement
   ...
endfor;
```

Operador Ternario

El operador ternario evalúa una condición y retorna un valor dependiendo si la condición es verdadera (true) o falsa (false).

Utilizando el operador ternario se simplifica de esta manera:

variable = (condición) ? valor-cuando-es-verdadera : valor-cuando-es-falsa;

Un ejemplo práctico:

```
<?php
// con un IF
if (date('G') < 12) {
$mensaje = 'Buenos días';
} else {
$mensaje = 'Buenas tardes';
}
print( $mensaje;)

// con el operador ternario
$mensaje = (date('G') < 12) ? 'Buenos días' : 'Buenas tardes';
print($mensaje);
?>
```

Funciones:

Definir subrutinas (funciones) en php es sencillo:

```
function prueba($a,$b) {
        $r=$a + $b;
        return $r;
}
```

Para invocar a la función basta con hacer $x=prueba(4,6);
Los parámetros que recibe la función pueden ser enteros, flotantes, strings, vectores u objetos es decir cualquiera de los tipos de datos soportado por PHP.
El valor devuelto por la función también puede ser cualquier tipo de datos de php.

Parámetros default:
Es posible asignar un valor default a los parámetros que recibe una función de forma tal que cuando se invoca la función si se ignora el parámetro el mismo es asignado al default.

```
function prueba($a=2,$b=3,$c=5) {
        //código
}
```

Si se llama prueba(4) // Entonces $a=4, $b=3 y $c=5

Se puede saber cuantos parámetros recibió una función usando func_num_args() y se puede obtener el iésimo parámetro de una función con func_get_arg(número_de_parámetro);
Finalmente los nombres de funciones pueden guardarse en variables e invocarse las mismas usando el

nombre guardado en una variable.

Ejemplo:
$nombre=sumar;
$nombre(4,5); //Llama a la función sumar

Esto permite guardar nombres de funciones en tablas, archivos, vectores etc lo cual da lugar a ciertas maniobras interesantes que no parece demasiado conveniente enumerar en este capítulo.

Parámetros dinámicos:
Consiste en definir una función sin parámetros y luego leer los parámetros pasados:

```
function prueba() {
        // código
}
```

Si de llama a la función de la siguiente manera:
prueba (2, “hola”);
podemos en su interior analizar los parámetros recibidos para actual en consecuencia:

```
function prueba() {
//implementación
// lectura delprimer parámetro
if (func_num_args() >= 1)
   { $num = func_get_arg(0);
   }
// lectura del segundo parámetro
if (func_num_args() >= 2)
   { $num = func_get_arg(1);
   }
// Código que hace algo
}
```

Funciones matemáticas.

Constantes predefinidas:

Tabla 1. Constantes Matemáticas

Constante	Valor	Descripción
M_PI	3.14159265358979323846	Pi
M_E	2.7182818284590452354	e
M_LOG2E	1.4426950408889634074	log_2 e
M_LOG10E	0.43429448190325182765	log_10 e
M_LN2	0.69314718055994530942	log_e 2
M_LN10	2.30258509299404568402	log_e 10
M_PI_2	1.57079632679489661923	pi/2
M_PI_4	0.78539816339744830962	pi/4
M_1_PI	0.31830988618379067154	1/pi
M_2_PI	0.63661977236758134308	2/pi
M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

Funciones:

Función	Descripción
var=abs(var);	Valor absoluto.
var=acos(var);	Arco coseno
var=atan(var);	Arco tangente
var=asin(var);	Arco seno
string=base_convert(string,base_from,base_to);	Realiza el pasaje de base correspondiente
var=bindec(string_binario);	Pasa de base 2 a decimal
int=ceil(float);	Redondea hacia arriba un número decimal (función techo)
var=cos(var);	Coseno
string=decbin(var);	Pasa de base 10 a base 2
string=dechex(var);	Pasa de base 10 a hexadecimal.
string=decoct(var);	Pasa de base 10 a octal
var=deg2rad(var);	Pasa de grados (degrees) a radianes
var=exp(var);	Devuelve e^var (función exponencial)
var=floor(var);	Redondea un número decimal hacia abajo (función piso)
var=hexdec(string);	Pasa de hexadecimal a base 10
var=log(var);	Logaritmo natural (base e)
var=log10(var);	Logaritmo en base 10
var=octdec(string);	Pasa de octal a base 10
var=max(array);	Devuelve el elemento máximo de un vector
var=max(var1,var2,...,varn);	Devuelve el elemento máximo
var=min(array);	Devuelve el elemento mínimo de un vector
var=min(var1,var2,.....,varn);	Devuelve el elemento mínimo
var=pow(base,exponente);	Devuelve base^exponente
var=rad2deg(var);	Convierte radianes en grados (degrees)
var=round(var);	Redondea un número no entero a su valor entero más cercano
var=rand(min,max);	Genera un número random entre los valores pasados
srand(var);	Inicializa la semilla del algoritmo de generación de números al azar. En general: srand((double)microtime()*1000000);
var=sin(var);	Función seno
var=tan(var);	Función tangente
var=sqrt(var);	Devuelve la raíz cuadrada de un número

Manejo de Cadenas de Caracteres (Strings)

A continuación se describe un resumen de las funciones mas importantes de PHP para manejo de strings.

Mayúsculas y minúsculas:

string=strtoupper(string);

Pasa un string a mayúsculas.

string=strtolower(string);
Pasa un string a minúsculas.

string=ucfirst(string);
Pasa a mayúscula el primer carácter de un string

string=ucwords(string);
Pasa a mayúsculas el primer carácter de cada palabra de un string (separadas por blancos, tabulaciones y saltos de línea)

Trimming:
string=chop(string);
Elimina blancos y saltos de línea a la derecha de un string dado.

string=ltrim(string);
Elimina blancos y saltos de línea a la izquierda de un string.

string=trim(string);
Elimina blancos y saltos de línea a derecha e izquierda de un string.

Comparaciones:

int=strpos(string1,caracter);
Devuelve la posición de la primera ocurrencia del caracer dentro de string1.

int=strspn(string1,string2);
Devuelve la longitud en caracteres de s1 contando desde el principio hasta que aparece un caracter en s1 que no esta en s2.

int=strcmp(string1,string2);
Compara dos strings y devuelve 1, 0 o -1 según sea mayor el primero, iguales o el segundo.

int=strcasecmp(string1,string2);
Idem anterior pero case-insesitive (no distingue mayúsculas y minúsculas)

int=strcspn(string1,string2);
Devuelve la longitud de s1 desde el principio hasta que aparece un caracter que pertenece a s2.

int=strstr(string1,caracter);
Devuelve todos los caracteres de s1 desde la primera ocurrencia de caracter hasta el final.

int=stristr(string1,string2);
Idem anterior pero case-insensitive (no distingue mayúsculas de minúsculas)

int=similar_text(string1,string2,referencia);
Analiza la semejanza entre dos strings, devuelve la cantidad de caracteres iguales en los dos strings, si se pasa como tercer parámetro una referencia a una variable devuelve en la misma el porcentaje de similitud entre ambos strings de acuerdo al algoritmo de Oliver (1993).

Ejemplo:
similar_text($st1,$st2,&$porcentaje);

Funciones de Parsing:

array=explode(separator,string);
Devuelve un vector donde cada elemento del vector es un substring del string pasado particionado de acuerdo a un cierto caracter separador.
Ejemplo:
$st="hola,mundo,como,estan"
$vec=explode(",",$st); //$vec=("hola","mundo","como","estan");

string=implode(separator,array);
Genera un string concatenando todos los elementos del vector pasado e intercalando separator entre ellos.

string=chunk_split(string,n,end);
end es opcional y por default es "\r\n", devuelve un string en donde cada "n" caracteres del string original se intercala el separador "end".

Ejemplo:
$st="hola mundo";
$st2=chunk_split($st,2,",");
//$st2="ho,la, m,un,do";

array=count_chars(string);
Devuelve un vector de 256 posiciones donde cada posición del vector indica la cantidad de veces que el caracter de dicho orden aparece en el string.

string=nl2br(string);
Devuelve un string en donde todos los saltos de línea se han reemplazado por el tag
 de html.

string=strip_tags(string,string_tags_validos);
Devuelve un string eliminando del string original todos los tags html, si se pasa el segundo parámetro opcional es posible especificar que tags no deben eliminarse (solo hace falta pasar los tags de apertura)

ejemplo:
$st2=strip_tags($st1,"
 <table>");
Elimina todos los tags html de $st1 excepto
 , <table> y </table>

string=metaphone(string);
Devuelve una representación metafónica (similar a soundex) del string de acuerdo a las reglas de pronunciación del idioma ingles.

string=strtok(string,separador);
Dado un separador obtiene el primer "token" de un string, sucesivas llamadas a strtok pasando solamente el separador devuelven los tokens en forma sucesiva o bien falso cuando ya no hay mas tokens.

Ejemplo:
$tok=strtok($st,"/");
while($tok) {
//Hacer algo
$tok=strtok("/");
}

parse_str(string);
Dado un string de la forma "nombre=valor&nombre2=valor2&nombre3=valor3", setea las variables correspondientes con los valores indicados, ejemplo:
parse_string("v1=hola&v2=mundo");
//Seteta $v1="hola" y $v2="mundo"

Codificación y decodificación ASCII.

char=chr(int);
Devuelve el caracter dado su número ascii.

int=ord(char);
Dado un caracter devuelve su código Ascii.

Caracteres Multibyte

string **mb_convert_encoding** (string `$str` , string `$to_encoding` [, mixed `$from_encoding`])

Convierte un string de un juego de caracteres a otro.

Ejemplo:

```
/* Convert ISO-8859-1 to UTF-8 */
$str = mb_convert_encoding($str, "UTF-8", "ISO-8859-1");
```

Substrings:

string=substr(string,offset,longitud);
Devuelve el substring correspondiente al string pasado desde la posición indicada por offset y de la longitud indicada como tercer parámetro, si no se pasa el tercer parámetro se toman todos los caracteres hasta el final del string.

string=substr_replace(string,string_reemplazo,offset, longitud);
Idem anterior pero el substring seleccionado es reemplazado por string_reemplazo, si string_reemplazo es "" entonces sirve para eliminar una porción de un string.

Búsquedas y Reemplazos.

str_replace(string1,string2,string3);
Reemplaza todas las ocurrencias de string1 en string3 por string2. Esta función no admite expresiones regulares como parámetros.

string=strtr(string1,string_from,string_to);
Reemplaza en string1 los caracteres en string_from por su equivalente en string_to (se supone que string_from y string_to son de la misma longitud, si no lo son los caracteres que sobran en el string mas largo se ignoran)

Ejemplo:
$st="hola mundo"
strtr($st,"aeiou","12345");
//$st="h4la m5nd4"

array=split(pattern,string);
Idem a explode pero el separador puede ser ahora una expresión regular.

boolean=ereg(pattern,string,regs);
Devuelve true o false según si el string matchea o no una expresión regular dada, el tercer parámetro es opcional y debe ser el nombre de un vector en donde se devolverán los matches de cada paréntesis de la expresion regular si es que la misma tiene paréntesis.

boolean=eregi(pattern,string,regs);
Idem anterior pero case-insensitive.

ereg_replace(pattern_from,string_to,string);
Reemplaza todas las ocurrencias de una expresión regular en string por el contenido de string_to.

eregi_replace(pattern_from,string_to,string);
Idem anterior pero no considera mayúsculas y minúsculas para la búsqueda de la expresión regular en el string.

Sintaxis básica de una expresión regular:

Los símbolos especiales "^" y "$" se usan para matchear el principio y el final de un string respectivamente. Por ejemplo:

"^el"	Matchea strings que empiezan con "el"
"colorin colorado$"	Matchea strings que terminan en "colorin colorado"
"^abc$"	String que empieza y termina en abc, es decir solo "abc" matchea
"abc"	Un string que contiene "abc" por ejemplo "abc" ,"gfabc", "algoabcfgeh", etc...

Los símbolos "*" , "+" y "?" denotan la cantidad de veces que un caracter o una secuencia de caracteres puede ocurrir. Y denotan 0 o más, una o más y cero o una ocurrencias respectivamente.

Por ejemplo:

"ab*"	Matchea strings que contienen una "a" seguida de cero o mas "b" Ej: "a", "ab", "cabbbb", etc
"ab+"	Matchea strings que contienen una "a" seguida de una o mas "b"
"ab?"	Matchea strings que contienen una "a" seguida o no de una "b" pero no mas de 1.
"a?b+$"	Matchea "a" seguida de una o mas "b" terminando el string.

Para indicar rangos de ocurrencias distintas pueden especificarse la cantidad máxima y mínima de ocurrencias usando llaves de la forma {min,max}

"ab{2}"	Una "a" seguida de exactamente 2 "b"
"ab{2,}"	Una "a" seguida de 2 o mas "b"
"ab{3,5}"	Una "a" seguida de 3 a 5 "b" ("abbb", "abbbb", "abbbbb")

Es obligatorio especificar el primer número del rango pero no el segundo. De esta forma + equivale a {1,}. * equivale a {0,} y ? equivale a {0,1}

Para cuantificar una secuencia de caracteres basta con ponerla entre paréntesis.

"a(bc)*"	Matchea una "a" seguida de cero o mas ocurrencias de "bc" ej: "abcbcbc"

El símbolo "|" funciona como operador "or"

"hola|Hola"	Matchea strings que contienen "hola" u "Hola"
"(b|cd)ef"	Strings que contienen "bef" o "cdef"
"(a|b)*c"	Secuencias de "a" o "b" y que termina en "c"

El carácter "." matchea a cualquier otro caracter.

"a.[0-9]"	Matchea "a" seguido de cualquier caracter y un dígito.
"^.{3}$"	Cualquier string de exactamente 3 caracteres.

Los corchetes se usan para indicar que caracteres son validos en una posición única del string.

"[ab]"	Matchea strings que contienen "a" o "b"
"[a-d]"	Matchea strings que contienen "a", "b" , "c" o "d"
"^[a-zA-Z]"	Strings que comienzan con una letra.
"[0-9]%"	Un dígito seguido de un signo %

También puede usarse una lista de caracteres que no se desean agregando el símbolo "^" dentro de los corchetes, no confundir con "^" afuera de los corchetes que matchea el principio de línea.

"[^abg]"	Strings que NO contienen "a" , "b" o "g"
"[^0-9]"	Strings que no contienen dígitos

Los caracteres "^.[$()|*+?{\" deben escaparse si forman parte de lo que se quiere buscar con una barra invertida adelante. Esto no es válido dentro de los corchetes donde todos los caracteres no tienen significado especial.
Ejemplos:
Validar una suma monetaria en formato: "10000.00", "10,000.00" .,"10000" o "10,000" es decir con o sin centavos y con o sin una coma separando tres dígitos.

^[1-9][0-9]*$

Esto valida cualquier número que no empieza con cero, lo malo es que "0" no pasa el test. Entonces:

^(0|[1-9][0-9]*)$

Un cero o cualquier número que no empieza con cero. Aceptemos también un posible signo menos delante.

^(0|-?[1-9][0-9]*)$

O sea cero o cualquier número con un posible signo "-" delante.

En realidad podemos admitir que un número empiece con cero para una cantidad monetaria y supongamos

que el signo "-" no tiene sentido, agreguemos la posibilidad de decimales:

^[0-9]+(\.[0-9]+)?$

Si viene un "." debe esta seguido de un dígito, 10 es válido pero "10." no Especifiquemos uno o dos dígitos decimales:

^[0-9]+(\.[0-9]{1,2})?$

Ahora tenemos que agregar las comas para separar de a miles.

^[0-9]{1,3}(,[0-9]{3})*(\.[0-9]{1,2})?$

O sea un conjunto de 1 a 3 dígitos seguido de uno más conjuntos de "," seguido de tres dígitos. Ahora hagamos que la coma sea opcional.

^([0-9]+|[0-9]{1,3}(,[0-9]{3})*)(\.[0-9]{1,2})?$

Y de esta forma podemos validar números con los 4 formatos validos en una sola expresión.

Manejo de fechas

PHP provee varias funciones para manipulación, validación y formateo de fechas, el formato interno para representar una fecha en PHP es el usado por Unix o sea una cantidad de segundos a partir de una fecha definida como EPOCH. Las funciones más importantes son las que describimos a continuación:

Date:

string=date(string_formato,time);

El segundo parámetro es opcional, si se pasa debe ser una fecha en formato de representación interna, si no se pasa se toma como fecha la fecha actual. El formato es un string de formato libre en el cual ciertos caracteres tienen un significado especial y son reemplazados por ciertos valores:

· a - "am" o "pm"
· A - "AM" o "PM"
· d - día del mes en dos dígitos con cero adelante si es necesario
· D - día de la semana en ingles en formato de tres letras Ej: "fri"
· F - Nombre del mes en ingles Ej: "January"
· h - Hora en formato de 12 horas: 01 a 12
· H - Hora en formato de 24 horas: 00 a 23
· g - Hora en formato de 12 horas sin ceros adelante: 1 a 12
· G - Hora en formato de 24 horas sinc ceros adelante: 0 a 23
· i - Minutos en dos dígitos: 00 a 59
· j - día del mes sin ceros adelante: 1 a 31
· l - día de la semana en ingles completo Ej: "Friday"
· L- Boolean que indica si el año es bisiesto (true=es, false=no)
· m - Número de mes 01 a 12
· n - Número de mes sin ceros adelante 1 a 12
· M - Nombre del mes en ingles en tres letras Ej: "Jan"
· s - Segundos 00 a 59
· S - Sufijo ordinal en ingles para el número de día (th,nd,st)
· t - Número de días para el mes actual 1 a 31
· U- Segundos pasados desde EPOCH (formato de representación interno)
· w - día de la semana en formato numérico (0=domingo)
· Y - Año en 4 dígitos
· y - Año en 2 dígitos
· z Día del año 1 a 365

Ejemplo:

date("Hoy es d/m/Y y la hora es: H:i:s");

Queda algo de la forma "Hoy es 10/05/2000 y la hora es 15:06:29"

Para obtener la representación interna de una fecha dado el día, mes, año, horas, minutos y segundos se usa la función mktime:

int=mktime (hora, minutos, segundos, mes, día, año)

Devuelve la cantidad de segundos pasados desde el epoch, luego puede usarse este valor devuelto como segundo parámetro de "Date" para formatear la fecha en el formato que se desee.

Otras funciones:

boolean=checkdate(mes, día, año)
Devuelve verdadero si la fecha pasada es válida. En caso contrario devuelve falso.

string=microtime()
Devuelve una representación de la hora actual incluyendo microsegundos, el string que devuelve tiene el formato "microsegundos segundos" y luego puede hacerse un explode del mismo tomando el espacio como separador para obtener los microsegundos.
La siguiente clase implementa timers con precisión de microsegundos que pueden usarse para medir duraciones de tiempo con gran precisión, lo cual es útil por ejemplo para realizar un "profile" de un script en php4 midiendo la duración de distintas partes del mismo (consultas a la base de datos, etc...)
(Veremos mejor la notación de la Programación Orientada a Objetos en PHP más adelante en este curso. Por ahora, vaya familiarizándose con la sintaxis)

```
// start('name') inicializa el timer con o sin nombre.
// stop('name') para el timer
// current('name') para el timer y devuelve el tiempo transcurrido
//
class Timer {
var $ss_timing_start_times;
var $ss_timing_stop_times;
function start($name='default'){
$this->ss_timing_start_times[$name]=explode(' ',microtime());
}
function stop($name='default'){
$this->ss_timing_stop_times[$name]=explode(' ',microtime());
}
function current($name='default') {
if(!isset($this->ss_timing_start_times[$name])){
return 0;
}
if(!isset($this->ss_timing_stop_times[$name])){
$stop_time=explode(' ',microtime());
} else {
$stop_time=$this->ss_timing_stop_times[$name];
}
$current=$stop_time[1]-$this->ss_timing_start_times[$name][1];
$current+=$stop_time[0]-$this->ss_timing_start_times[$name][0];
return $current;
}
}
?>
```

Ejemplo de uso:

```
$tim=new Timer();
$tim->start();
//codigo.
$tiempo=$tim->current();
```

Si se quieren usar múltiples timers simultáneos puede pasarse un "nombre" de timer a las funciones start, stop y current para diferenciar distintos timers y usarlos en forma independiente.

Conceptos básicos en la generación de sitios web dinámicos con PHP: layout

Una característica interesante de php es que permite realizar “includes” dentro de un script php, de esta forma se puede modularizar una página o el layout de una página en varios módulos php que se desarrollan en forma independiente, además pueden desarrollarse en php componentes reusables que luego se utilizan en otras páginas o incluso en otros sites.
Una forma común de trabajo usando php para generar sitios dinámicos es definir “templates” o “layouts” en los cuales se divide la página en “zonas” o “módulos” que serán desarrollados en php, el layout de la página con la forma y tamaño de cada zona se puede definir sin problemas usando por ejemplo tablas de html. A lo largo de este curso desarrollaremos todos los ejercicios y aplicaciones de los temas a medida que se estudian distintas características de php usando el concepto de layout. Supongamos que tenemos por el momento un único “template” o “layout” para nuestro sitio que determina la forma en la cual se vera la “home page” del mismo, el equipo de diseño nos entrega el siguiente layout:

<table>
<tr><td>LOGO</td><td colspan="3">BANNER</td></tr>
<tr><td colspan="4">BUSCADOR</td></tr>
<tr><td>BOTON S1</td><td>BOTON S2</td><td>BOTON S3</td><td>BOTON S4</td></tr>
<tr><td colspan="2">Barra de links y
Aplicaciones</td><td colspan="2">CONTENIDOS
.
.
.
.
.
.
.
.
CONTENIDOS</td></tr>
<tr><td colspan="4">Información de copyrigh, y pie de página</td></tr>
</table>

Como podemos ver el site cuenta con 4 secciones que se acceden desde una barra navegadora ubicada debajo del search box, además existe una barra de links y servicios a la izquierda y una zona de contenidos que es la zona principal de la página.
Aun sin saber que funcionalidad tiene o de que forma se debe implementar cada parte podemos esquematizar el layout de la página usando php y html de la siguiente forma:

```
<?php print('<?xml version="1.0" encoding="UTF-8"?>'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Layout</title>
</head>
<body>
<table width="640" border="1">
<tr><td width="20%">Logo</td><td colspan="3">Banner</td></tr>
```

```
<tr><td colspan="4">buscador</td></tr>
<tr><td>s1</td><td>s2</td><td>s3</td><td>s4</td></tr>
<tr><td width="20%">ColIzq</td><td colspan="3">Contenidos</td></tr>
</table>
</body>
</html>
```

Luego podemos reemplazar cada “zona” de la home page por un include en php que generara dinámicamente la parte de la página en cuestión:

```
<?php print('<?xml version="1.0" encoding="UTF-8"?>'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Layout</title>
</head>
<body>
<table width="640" border="1">
<tr><td width="20%"><? include(“logo.php”);?></td><td
colspan="3"><? include(“banner.php”);?></td></tr>
<tr><td colspan="4"><? include(“buscador.php”);?></td></tr>
<tr><? include(“botonera.php”);?></tr>
<tr><td width="20%"><? include(“izq.php”);?></td><td
colspan="3"><? include(“contenidos_home.php”);?></td></tr>
</table>
</body>
</html>
```

De esta forma hemos modularizado el layout de la página y tenemos como resultado que deben desarrollarse
los siguientes módulos:

· logo.php
· banner.php
· buscador.php
· botonera.php
· izq.php
· contenidos_home.php